# A Fully Portable TCP Implementation Using XFSMs

Giuseppe Bianchi[1,2], Michael Welzl[3], Angelo Tulumello[1,2], Giacomo Belocchi[1,2], Marco Faltelli[1,2], Salvatore Pontarelli[1]
[1]Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy.
[2]University of Rome Tor Vergata, Italy.
[3]Department of Informatics, University of Oslo, Norway.

## ABSTRACT

XTRA (XFSM for TRAnsport) is a first step towards "code-once-port-everywhere" transport protocols. XTRA's platform-agnostic programming abstraction, based on an extended finite state machine formalization of a desired transport layer task, is amenable not only to SW engines, but can be directly executed in CPU-less custom HW, thus permits to harness FPGA-based NICs' offloading opportunities without any re-coding effort. We experimentally demonstrate that XTRA enables us to port a customized TCP implementation across three completely different environments (HW proof-of-concept on a NetFPGA board, User-space SW over Linux' Open Data Plane, and NS3 emulator).

## CCS CONCEPTS

• **Networks → Transport protocols**; • **Hardware → Networking hardware**;

## KEYWORDS

TCP, Offload, Data Plane Programmability, Hardware Acceleration

**ACM Reference format:**
Giuseppe Bianchi[1,2], Michael Welzl[3], Angelo Tulumello[1,2], Giacomo Belocchi[1,2], Marco Faltelli[1,2], Salvatore Pontarelli[1] [1]Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy. [2]University of Rome Tor Vergata, Italy. [3]Department of Informatics, University of Oslo, Norway. . 2018. A Fully Portable TCP Implementation Using XFSMs. In *Proceedings of ACM SIGCOMM 2018 Conference Posters and Demos, Budapest, Hungary, August 20–25, 2018 (SIGCOMM Posters and Demos '18),* 3 pages.
https://doi.org/10.1145/3234200.3234237

## 1 INTRODUCTION

The question addressed in this demo is the following. Is there some convenient "*code-once-port-everywhere*" platform-independent programming abstraction sufficiently complete to write a transport layer function, or even a full-fledged transport protocol, and at the same time capable to make such an abstract description of the transport protocol portable across *both* SW and HW platforms (e.g. FPGA-based NICs)[4, 5]?


*SIGCOMM Posters and Demos '18, August 20–25, 2018, Budapest, Hungary*

ACM ISBN 978-1-4503-5915-3/18/08…$15.00
https://doi.org/10.1145/3234200.3234237

So far, portability of network functions (also) across HW platforms has been mainly addressed via HW-amenable abstractions such as OpenFlow [2] or P4 [3], and has been restricted to stateless, packet-based, processing tasks. Even stateful programming abstractions such as OpenState [1] are far from being able to support complex transport layer tasks involving buffering or scheduling—not to mention a TCP implementation analogous to those deployed in end terminals' Operating Systems.

Quite interestingly, even portability "only" across SW platforms has been somewhat disregarded to date. Indeed, the transport layer is rich with protocols that are hand-customized for various use cases (TCP, UDP, MPTCP, SCTP, Adobe's RTMFP, QUIC, to name but a few). The only thing that seems to be interchangeable in some cases is the congestion control algorithm (Linux and FreeBSD have "pluggable congestion control" frameworks), such that congestion controls are also separately customized (PCC, Remy, BBR, Cubic, Verus, Sprout, DCTCP, DCQCN, to name but a few). All of these implementations are silos, they need to be re-done when porting code, and efficient hardware usage requires deep customization per system. In line with our idea, HotCocoa [7] and the "Congestion Control Plane" (CCP) [8] make congestion control deployment more flexible and efficient (the latter takes them entirely out of the data path), but both of these proposals are strictly designed to support congestion control only, while our focus is on a complete protocol.

## 2 INTRODUCING XTRA

What would it take to make the transport layer *truly* programmable—such that transport protocols could be implemented *once*, in *one* appropriate representation, and run *everywhere*? We found that suitably extended Finite State Machines (XFSMs), already proven successful in stateful flow processing tasks if equipped with technical means and primitives to describe and manage time, buffer space, packet generation, and arithmetic operations, are a good abstraction model to also formalize transport protocols. The advantage of XFSMs stems from their simplicity and amenability to *both* hardware and software implementation. In a software implementation, the use of XFSMs adds only a little overhead with respect to the normal operations of standard programming languages. Moreover, being XFSMs Turing complete [9], as long as the actions implemented support an adequate set of primitives, the abstraction does not limit the computational possibilities. We developed an XFSM execution engine "XTRA" (XFSM for TRAnsport) which offers an abstraction to define transport protocols which makes them entirely platform independent while being efficient to execute. Having ported XTRA to three extremely different platforms (OpenDataPlane SW, FPGA HW, NS3 emulator), we demonstrate its benefits via a fully portable XFSM-based TCP implementation. The demonstration will show

that our TCP implementation can run on all of these platforms without changing a single line of TCP code. The main elements of XTRA are summarized below:

**Small set of elementary and independent "bricks"**. This first requirement is somewhat straightforward and largely accepted. Many successful network programming technologies build upon the idea of composing a comprehensive network operation via elementary bricks such as basic and mutually independent building blocks.

**Explicit modeling of a stateful operation**. One of the key elements of XTRA is that it explicitly provides an upfront way to formally describe a non-trivial state evolution such as that of a TCP connection, thus involving many packets, buffer information, and comprising several state parameters continuously updated at runtime (RTT estimate, congestion window, etc.).

**Time management**. Transport protocols require the management of complex events involving time, such as triggering a stored packet transmission at a later time, as well as to support primitives as retransmissions or pacing; XTRA provides a flexible interface for programming the behavior associated to time management.

## 3 TIMER-BASED TCP

TCP's connection (Listen, SYN-Sent, SYN-Received, etc.) and congestion control (Slow Start, Congestion Avoidance, Fast Retransmit / Fast Recovery) state machines as well as the Reno congestion window update rules can be readily transformed into an XFSM. The Fast Recovery phase itself, however, is exceedingly complex. To address this, in this demo, we focused on a strictly timer-based version of TCP, which we labeled as *Timer-Based TCP (TB-TCP)*. This choice follows the intuition of RACK ("Recent ACKnowledgment") [6], which significantly changes the underlying logic of TCP's loss recovery: it uses the notion of *time* instead of counting segments or byte sequences to detect losses and decide to retransmit segments. When an ACK arrives that acknowledges some but not all transmitted segments, any segments that were sent earlier (by a minimum time called "reordering window") than the acknowledged segments are assumed to be lost. Not only does this allow RACK to detect more losses than other algorithms, it can effectively replace other algorithms make Fast Recovery much more complex.

While RACK relies on (SACK) information from DupACKs, it seems obvious that the idea of considering the amount of time that has expired since a segment was sent could also be implemented more directly, by using a physical timer (which is efficient to implement for an XFSM) for every transmitted segment. Indeed, the authors of RACK state that it "conceptually arms a (virtual) timer on every packet sent" [6]—we instead arm a physical timer on every packet sent.

Using the ns-3 implementation with emulation to send traffic across a real testbed, we tested the efficiency of the loss recovery phase by considering the flow completion time of the slowest of six parallel flows (a common number used by browsers) in approx. 400 experiments covering a large range of network conditions (varying the bottleneck capacity, RTT and the length of the bottleneck queue); Figure 1 shows that the performance of TB-TCP is very close to standard TCP, and in some cases it even outperforms both the Linux and FreeBSD TCP implementation.

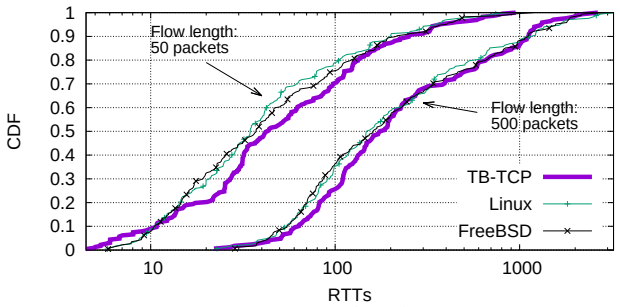


**Figure 1: Flow Completion Time (FCT) in RTTs of the slowest of six flows**

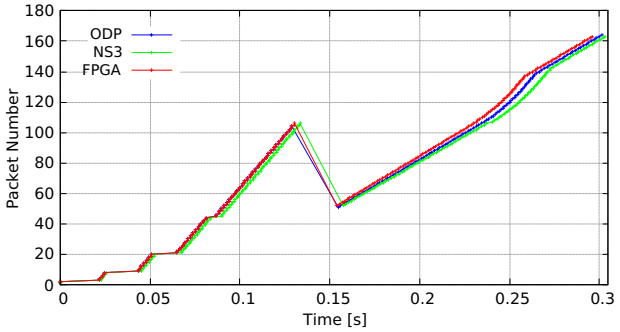


**Figure 2: Time-sequence diagram of a TB-TCP flow beginning in Slow Start, then losing a packet, and concluding with our timer-based Fast Recovery phase.**

## 4 DEMONSTRATING PORTABILITY

Our demonstration will show that: 1) TB-TCP runs on a pure hardware NetFPGA implementation of XTRA, 2) the same TB-TCP code runs on an ns-3 implementation of XTRA, and 3) the same TB-TCP code runs on an ODP implementation of XTRA. We will let audience members propose code changes to witness how they equally affect the implementation on two of these platforms. A video of the demonstration showing the XTRA language, its mapping on the three implementations and some examples of the collected results obtained running the TB-TCP protocol on the three implementation is available at the following link: https://www.dropbox.com/s/1u5njq0ixbu37b6/xtra-demo.mp4?dl=0.

The time-sequence diagram in Figure 2 visualizes just how close the behavior of the three implementations is. This test was carried out using a single sender and receiver, interconnected with a physical link and using linux Traffic Control (TC) *netem* module. The parameters used were: bandwidth of 6Mbps, queue length of 20 packets and one-way delay of 10ms. We can also see minor deviations between the three implementations, which are caused by small timing differences in how packets are clocked out. For instance, ns-3 is slightly slower at transmitting packets using emulation. The hardware implementation appears to be a little faster, which is due to the lower precision for divisions.

For future work, we are planning to investigate porting other protocols such as SCTP or QUIC to XTRA.

## ACKNOWLEDGMENTS

This work has been partly funded by the EU commission in the context of the 5G-PICTURE project, Grant Agreement 762057.

## REFERENCES


[1] G. Bianchi, M. Bonola, A. Capone, C. Cascone, "OpenState: programming platform-independent stateful openflow applications inside the switch", ACM SIGCOMM Comp. Commun. Rev. 44(2), 2014

[2] N. McKeown et. al. "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Comp. Commun. Rev. 38(2), 2008

[3] P. Bosshart et. al. "P4: Programming protocol-independent packet processors", ACM SIGCOMM Comp. Commun. Rev. 44(3), 2014

[4] P. Shinde, A. Kaufmann, T. Roscoe, S. Kaestle, "We Need to Talk About NICs", 14th USENIX HotOS 2013

[5] Y. Le et. al. "UNO: Uniflying Host and Smart NIC Offload for Flexible Packet Processing, 2017 Symp. on Cloud Comput. (SoCC), 2017

[6] Y. Cheng, N. Cardwell, N. Dukkipati, "RACK: a time-based fast loss recovery", 98th IETF Meeting, March 2017

[7] M.T.Arashloo, M.Ghobadi, J.Rexford, D.Walker, "HotCocoa: Hardware Congestion Control Abstractions", Proc. 16th ACM Workshop on Hot Topics in Networks (HotNets), 2017.

[8] Akshay Narayan, Frank Cangialosi, Prateesh Goyal, Srinivas Narayana, Mohammad Alizadeh, Hari Balakrishnan, "The Case for Moving Congestion Control Out of the Datapath", Proc. 16th ACM Workshop on Hot Topics in Networks (HotNets), 2017.

[9] C.-J. Wang and M. T. Liu. A Test Suite Generation Method for ExtendedFinite State Machines Using Axiomatic Semantics Approach. InProc.of the XII IFIP TC6/WG6.1 Int.l Symp. on Protocol Specification, Testingand Verification, pages 29-43, 1992.